Tampereen yliopisto

Max Mecklin

AUDION JA VIDEON TOISTAMINEN VERKOSSA TAHDISTETUSTI MONELLA LAITTEELLA

Informaatioteknologian ja viestinnän tiedekunta
Kandidaatintyö
Tammikuu 2020

TIIVISTELMÄ

Max Mecklin: Audion ja videon toistaminen verkossa tahdistetusti monella laitteella
Kandidaatintyö
Tampereen yliopisto
Tieto- ja sähkötekniikan tutkinto-ohjelma
Tammikuu 2020

Tässä työssä tutkitaan suoratoistoprotokollia ja miten niillä suoratoistettua sisältöä voitaisiin tahdistaa. Lisäksi tutkitaan, miten voidaan tahdistaa paikallisesti toistettua sisältöä useammalla laitteella. Tahdistettua toistamista usealla laitteella voitaisiin käyttää pienissä tapahtumissa, missä ei ole massiivista budjettia toteuttamaan sen audiovisuaalisia tarpeita.

Suoratoistoprotokollista valittiin Real Time Streaming Protocol (RTSP), jota käytetään käytännön testaukseen. Testaukseen toteutetaan palvelin- ja asiakasohjelma C-ohjelmointikielellä. Lisäksi käytetään GStreamer-kirjastoa, joka tarjoaa suoratoiston työkaluja. Toteutuksella suoritetaan testejä, joista selviää sen toimivuus langallisissa ja langattomissa lähiverkoissa (LAN ja WLAN). Lisäksi toteutuksen asiakasohjelmaa verrataan VideoLAN Client -mediasoittimeen (VLC), joka sisältää RTSP-asiakasohjelman.

Testeissä käytetään kahta kannettavaa tietokonetta, kameraa ja palvelinta. Tietokoneet suoratoistavat testivideon palvelimelta ja kameralla videoidaan tietokoneiden näyttöjä videolle, josta voidaan 1/60 sekunnin tarkkuudella laskea kannettavien välisen suoratoiston aikaero. Testeistä selviää, että VLC on huomattavasti parempi asiakasohjelma, kuin testitoteutuksessa määritelty. LAN-verkko on parempi suoratoiston tahdistamisen kannalta kuin WLAN-verkko, sillä WLAN-verkoissa on enemmän häiriöitä, jotka johtavat ylimääräiseen viiveeseen.

Avainsanat: suoratoisto, tahdistus, video, tietoverkko

Tämän julkaisun alkuperäisyys on tarkastettu Turnitin OriginalityCheck -ohjelmalla.

SISÄLLYSLUETTELO

LYHENTEET JA MERKINNÄT

CC	CSRC count
CSRC	Contributing source
DASH	Dynamic Adaptive Streaming over HTTP
GPS	Global Positioning System
HLS	HTTP Live Streaming
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
LAN	Local Area Network
MPD	Media Presentation Description
M	Marker
NTP	Network Time Protocol
PT	Payload type
P	Padding
RTCP	RTP Control Protocol
RTMP	Real Time Messaging Protocol
RTP	Real-time Transport Protocol
RTSP	Real Time Streaming Protocol
SSRC	Synchronization source
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
UTC	Koordinoitu yleisaika
VLC	VideoLAN Client
V	Version
WebRTC	Web Real-time communication
WLAN	Wireless Local Area Network
XML	Extensible Markup Language
X	Extension

1 JOHDANTO

Nykymaailmassa järjestetään paljon erikokoisia tapahtumia, jotka vaativat omanlaisiaan audiovisuaalisia toteutuksia. Nämä toteutukset ovat yleensä massiivisia, monimutkaisia ja vaativat joukon ammattilaisia pystytykseen ja purkuun. Tahdistus on niissä tärkeää, sillä muuten varsinkin audio voi häiritä kokemusta, jos sen kuulee eri aikaan eri paikoista tai jos vierekkäiset videot kulkevat eri tahdissa. Tietokoneiden ja tietoverkkojen kehitys laskentatehollisesti ja kapasiteetillisesti avaa ihmisille paljon tilaa toteuttaa muun muassa omia melko reaaliaikaisia lähetyksiä internetin yli tuhansille muille ihmisille katsottavaksi.

Tietoverkon yli median toistamiseen on kehitetty suoratoistoprotokollia, joiden avulla voidaan toistaa sisältöä reaaliaikaisesti. Näissä protokollissa on myös ominaisuuksia tahdistaa useammasta lähteestä tulevia video- ja audiovirtoja keskenään. Onko niitä ominaisuuksia mahdollista käyttää vastaanottajien tahdistukseen? Voitaisiinko tietokoneilla ja tietoverkoilla toteuttaa audiovisuaalinen järjestelmä, jolla sisältö voitaisiin toistaa tahdistetusti? Tässä työssä tutkitaan eri tapoja toteuttaa tahdistettuja audiovisuaalisia toistamisjärjestelmiä hyödyntäen tietoverkkoja, sekä toteutetaan yksi niistä ja testataan sitä. Lisäksi tutkitaan paikallisen toiston tahdistusta usealla laitteella. Testauksen tulokset esitellään ja niiden avulla analysoidaan, miten olisi mahdollista kehittää paras mahdollinen toteutus.

Luvussa 2 käydään läpi, millaisilla tavoilla järjestelmän tiedonsiirto ja tahdistus on mahdollista toteuttaa ja luvussa 3 esitetään esimerkkitoteutus. Luvussa 4 esitetään testausmenetelmä ja analysoidaan sen tulokset. Lopuksi luku 5 on yhteenveto.

2 TIEDONSIIRTO JA TAHDISTETUSTI TOISTAMINEN

Tässä luvussa esitettään tapoja toistaa sisältöä tahdistetusti. Luvussa 2.1 esitellään suoratoistoprotokollia ja 2.2 esitellään paikallisen toiston tahdistusta.

Audion ja videon toistaminen samanaikaisesti usealla laitteella on toteutettavissa monella tavalla. Suoratoistaminen tai tiedostojen toisto paikallisesti ovat kaksi merkittävästi toisistaan eroavaa tapaa. Tahdistus ei ole kuitenkaan täydellistä, vaikka laitettaisiin laitteet toistamaan paikallista sisältöä oman kellon mukaan, sillä laitteiden kellonajat saattavat vaihdella. Myös laitteiden välisessä tiedonsiirrossa saattaa ilmetä tietoverkkojen aiheuttamia arvaamattoman pituisia viiveitä suoratoistopalvelusta toistettaessa. Tällöin tarvitaan toteutus, joka on puhtaasti tarkoitettu tahdistettuun toistoon.

2.1 Suoratoisto

Suoratoisto on tapa toistaa audiota ja videota toiselta laitteelta, josta sitä lähetetään reaaliaikaisesti tai jonne se on etukäteen tallennettu. Suoratoiston olennaisin ominaisuus on se, että käyttäjän ei tarvitse tallentaa kokonaisia tiedostoja omalle laitteelleen. Suoratoistoprotokollat lataavat muistiin vain tarvittavan segmentin tiedostosta ja poistavat sen käytön jälkeen. Lähetettävä data pakataan verkon kuormituksen minimoimiseksi. [1]

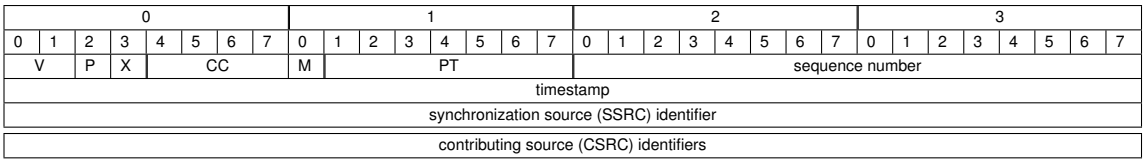
Suoratoiston ja tietoverkkojen teknologisen kehityksen myötä muun muassa moni videovuokraamoyritys on siirtänyt toimintansa internetiin. Internetissä palvelu keskittyy myymään suoratoistettavaa sisältöä kuukausimaksuja vastaan. Tällaisia yrityksiä ovat muun muassa Netflix ja HBO. Myös YouTube on tunnettu suoratoistopalvelu, mutta sen sisältö on pääasiassa ilmaista. YouTuben tuottajina toimivat käyttäjät itse. Se ansaitsee rahansa myymällä mainospaikkoja videoista muille yrityksille ja myymällä kuluttajille YouTube Premium -jäsenyyksiä.

2.1.1 Real-time Transport Protocol (RTP)

RTP eli Real-time Transport Protocol tarjoaa päästä päähän -kuljetuspalvelun datalle reaaliaikaisominaisuuksin, kuten interaktiivisen audion ja videon. RTP:hen kuuluu järjestysnumerointi, aikaleimat ja kuljetusmonitorointi. Sovellukset käyttävät yleensä RTP:tä UDP-

protokollan päällä hyödyntääkseen UDP:n kanavointi- ja tarkistussummaominaisuuksia. RTP itsessään ei tarjoa ajoitettua toimitusta tai muita palvelulaadun ominaisuuksia, vaan luottaa siihen, että alempi taso tarjoaa ne. Järjestysnumeroiden avulla vastaanottaja voi järjestää paketit oikeaan järjestykseen. RTP on pääasiassa suunniteltu videoneuvotteluihin, joissa on paljon osallistujia. RTP:tä voidaan soveltaa myös muihin käyttötarkoituksiin. [2]

RTP koostuu kahdesta aliprotokollasta, RTP Data Transfer Protocol ja RTP Control Protocol (RTCP). RTCP hoitaa vastaanottajien palautteita ja sen mukaan korjaa ongelmia suoratoistossa. [2] RTP Data Transfer -protokollan vakio-otsikko löytyy kuvasta 2.1 ja kenttien selityksen löytyvät taulukosta 2.1.



Kuva 2.1. *RTP-protokollan datapakettien vakio-otsikko. Nämä kentät esiintyvät kaikissa datapaketeissa. Muokattu lähteestä [2, s. 12].*

V	Merkitään käytetty RTP-versio.
P	Jos bitti on 1, on käytössä yksi tai useampi oktetti täytettä lopussa.
X	Jos bitti on 1, vakio-otsikon perässä on yksi lisäotsikko.
CC	CSRC-tunnisteiden määrä. Tunnisteet sijaitsevat vakio-otsikon lopussa.
M	Vapaaehtoinen merkkibitti.
PT	RTP-paketin sisällön tyyppi.
sequence number	Järjestysnumero, jota käytetään pakettien järjestelyyn ja puuttuvien pakettien tunnistamiseen.
timestamp	Paketin aikaleima.
SSRC	Lähteen tahdistustunniste.
CSRC	Sisältöön osallistuneiden tunnisteet. Tunnisteita voi olla 0–15.

Taulukko 2.1. *Otsikon kenttien lyhenteet ja niiden määritelmät, kuvasta 2.1.*

RTP:n Data Transfer -protokollalla on Network Time Protocol:aan (NTP) yhteensopivat aikaleimat, joiden avulla voidaan tahdistaa useasta lähteestä tulevat audio- ja videovirrat. Lähteiden tahdistus vaatii NTP:n käyttöä kaikissa lähettävissä laitteissa, jolloin laitteiden kellonajat ovat samat. CSRC-tunnisteita voidaan videoneuvotteluissa käyttää ilmaisemaan, kuka puhuu. [2] NTP:tä käsitellään lisää luvussa 2.2.

RTCP perustuu määräajoin toistuviin ohjauspaketteihin, jotka lähetetään kaikille istunnon osallistujille. Nämä paketit käyttävät samaa jakomekanismia kuin datapaketit. RTCP:llä kuljetetaan lähettäjä- ja vastanottajaraportteja, joissa välitetään laatupalautetta. Raportit sisältävät myös NTP- ja RTP-aikaleimoja, joiden avulla voidaan laskea pakettien kierrosaikoja, eli kuinka kauan vastaanottajalla kestää vastaanottaa paketti ja kuinka kauan

vastauksen saaminen kestää. [2]

Web Real-time communication (WebRTC) on Googlen kehittämä avoimen lähdekoodin protokolla. WebRTC käyttää RTP-protokollaa tiedonsiirtoon. WebRTC:n tarkoitus on mahdollistaa videoneuvottelun verkon yli lyhyellä viiveellä. [3] Toteutus tulisi olla verkkoselainpohjainen, mikä voi tuoda suorituskykyongelmia, sillä verkkoselainten kautta tietokoneiden resurssien saanti on rajoitetumpaa [4].

2.1.2 Real Time Streaming Protocol (RTSP)

RTSP eli Real Time Streaming Protocol (versio 2) on sovellustason protokolla, jota käytetään tyypillisesti median suoratoistoon. Se siis valmistelee ja ohjaa tiedonsiirtoa, jolla on reaaliaikaisia ominaisuuksia. Se toimii pääasiassa palvelimien ja asiakkaiden välillä, mutta tukee myös välityspalvelimia palvelimien ja asiakkaiden välillä. Asiakas voi pyytää palvelimelta tietoa suoratoistettavasta mediasta ja sen jälkeen pyytää sitä toistamaan, pysäyttämään tai lopettamaan toistaminen kokonaan. Pyydetty media voi olla useampia audio- ja videovirtoja, jotka on toimitettu aikatahdistettuna palvelimilta asiakkaille. [5]

RTSP on kaksisuuntainen pyyntö- ja vastausprotokolla, joka ensiksi perustaa kontekstin ja sitten ohjaa sisällön palveluntarjoajalta kuluttajalle. RTSP käyttää tekstipohjaisia viestejä, pyyntöjä ja vastauksia, jotka saattavat sisältää binääriosioita. RTSP vaatii että palvelin ja asiakas käyttävät TCP:tä ja TLS:ää TCP:n yli sen viestien kuljetukseen. RTSP:n median kuljetukseen voidaan käyttää esimerkiksi RTP:tä tai TCP:tä. [5]

Kun RTSP-istunto on saatu käyntiin, asiakas voi aloittaa median kuljetuksen ohjauksen. Asiakas voi aloittaa toistamisen haluamastaan kohdasta tai pyytää sen pysäytystä. Tarkennettu toistaminen voidaan mahdollistaa Range-nimisellä otsikkokentällä. Siihen voidaan kertoa aloitus- ja lopetusajat. [5]

2.1.3 Real Time Messaging Protocol (RTMP)

RTMP eli Real Time Messaging Protocol on Adoben kehittämä protokolla realiaikaiseen viestintään, kuten videon ja audion lähettämiseen. RTMP:ssä kulkee aikaleimat lähetettävän tiedon mukana, josta vastaanottajat tietävät, milloin sisältö pitää toistaa/näyttää. RTMP sopii moneen eri käyttötarkoitukseen, sillä voidaan lähettää tietoa yhdelle tai useammalle vastaanottajalle. [6]

RTMP:n viestissä on kaksi osaa, otsikko ja data. Otsakkeesta löytyy muun muassa viestin tyyppi, datan pituus ja aikaleima. Data sisältää audion tai videon segmentin. Viestien tyypit ovat nimeltään *User Control Messages*, *Audio Messages*, *Video Messages*, *Command Messages*, *Shared Object Messages* ja *Data Messages*. [6]

2.1.4 HTTP Live Streaming (HLS)

HLS eli HTTP Live Streaming -protokolla on HTTP:n päälle toteutettu suoratoistoprotokolla. Sitä käytettäessä vastaanottaja voi vaihdella virran bittinopeutta yhteyden mukaan, jotta suoratoistoa voidaan ylläpitää ilman katkaisuja ja parhaalla mahdollisella laadulla. Se tukee myös HTTP-välimuistin käyttämistä isojen vastaanottajamäärien palvelemiseksi. [7]

Tiedonsiirto perustuu soittolistoihin, jotka ovat UTF-8-merkistöä noudattavia tekstitiedostoja. Toistolistat sisältävät tiedot toistettavista mediasegmenteistä, jotka laite lataa ja toistaa oikeassa järjestyksessä. Lataus tulee suorittaa HTTP:n yli. Kun laite on saanut toistettua kaiken, lataa se toistolistan uudestaa, löytääkseen lisää mediasegmenttejä. Pääsoittolistan avulla, voidaan tarjota laitteelle useita toistolistoja, samalla sisällöllä mutta eri bittinopeudella. [7]

2.1.5 Dynamic Adaptive Streaming over HTTP (DASH)

DASH eli Dynamic Adaptive Streaming over HTTP määrittää XML- (Extensible Markup Language) ja binääriformaatteja, joiden avulla voidaan suoratoistaa mediasisältöä tavallisilta HTTP-palvelimilta tavallisille HTTP-asiakasohjelmille. Pääformaatteja on kaksi, Media Presentation Description (MPD) ja itse toistettava sisältö. MPD on XML-tiedosto, joka sisältää tiedot suoratoistosta. Toistettava sisältö on binääriä. Mediasisältö kuljetetaan segmenteissä, joissa yksi segmentti voi sisältää esimerkiksi useammankielisen audioraidan tai videota useammasta kamerasta. [8]

HTTP-protokollan käyttö suoratoistoprotokollissa mahdollistaa sen ominaisuuksien hyödyntämisen, kuten uudelleenohjauksen, välimuistin ja todentamisen. HTTP-protokollan yhteydessä voi myös käyttää vaivattomasti TLS-salausta. [8]

2.2 Paikallinen toisto NTP:llä tahdistaen

Kun audio tai video on ennalta tunnettua, se voidaan ladata toistolaitteille etukäteen, jolloin tietoverkko ei aiheuta viivettä tiedonsiirrossa. Kun tietoa ei siirretä reaaliajassa, voidaan se siirtää tavannomaisin keinoin laitteille, kuten TCP-yhteyden yli. Tällöin ei tarvitse huolehtia tahdistamisesta tässä vaiheessa. Sisältö pitää kuitenkin toistaa tahdistetusti muitten laitteiden kanssa. Tahdistusta voidaan toteuttaa tekemällä yhdestä laitteesta isäntä, joka antaa komentoja muille laitteille. Komentojen tulisi sisältää tiedot toistettavasta mediatiedostosta ja sen toistoajanhetkestä. Ajanhetken voisi komennossa ilmaista toistojärjestyksellä tai aikaleimoilla. Isännän tulisi myös laskea ajanhetket, jolloin toistettava sisältö vaihtuu.

Eräs paikallisen toiston järjestelmä on toteutettu Päivölän Opiston matematiikkalinjalla,

mutta sen toteutuksessa ei ole keskitytty tahdistukseen. Matematiikkalinjan toteutus vaatii paljon valmistelua, jotta toistettava sisältö saadaan jokaiselle laitteelle ja vaatii vähintää yhden ihmisen ylläpidon käytössä.

Suoratoistoon verrattuna paikallinen toisto ei vaadi suurta tiedonsiirtoa tietoverkon yli, jolloin sen voi toteuttaa pienemmän kapasiteetin tietoverkkoihin. Paikallisessa toistossa ei välttämättä tarvita tietoverkkoa lainkaan aikakriittisesti, sillä sisältö on jo laitteilla ja toistokomennot voivat antaa aikaleiman toistoajanhetkestä, jolloin se voidaan toimittaa ennen toistoa. Paikallinen toisto on myös oletettavasti luotettavampi langattomassa tietoverkossa kuin suoratoisto. Suoratoisto toisaalta helpottaa toistettavan sisällön vaihtoa. Suoratoistolla voidaan myös korjata tahdistusta tarkemmin toiston aikana.

Mikäli laitteille annetaan toistoajankohdan sisältävä komento tietoverkon yli etukäteen, täytyy myös varmistaa, että laitteiden paikalliset kellot ovat samassa ajassa. Tähän sopii ratkaisuksi NTP. NTP:tä käyttäen voidaan kysyä useammalta muulta laitteelta viittausta kellonaikaan, jonka avulla saadaan tarkempi kellonaika [9]. Tätä protokollaa käyttäen, samoilla viittauksilla, voidaan saada laitteiden kellot tarpeeksi lähelle samaa kellonaikaa.

NTP-palvelimia on kahdenlaisia, pääpalvelimia ja toissijaispalvelimia. Pääpalvelimet tahdistuvat viitekelloon, jota voidaan suoraan jäljittää UTC:hen. Viitekellon voi tarjota esimerkiksi GPS, Galileo tai muu paikannusjärjestelmä. Asiakasohjelma tahdistuu yhteen tai useampaan paluusuunnan palvelimeen, mutta ei tarjoa tahdistusta riippuvaisille asiakasohjelmille. Toissijaispalvelimilla on yksi tai useampi paluusuunnan palvelin ja yksi tai useampi tulosuunnan palvelin tai asiakasohjelma. NTP-protokolla käyttää palvelimissaan ja asiakasohjelmissaan erinäisiä algoritmeja laskeakseen "oikean" ajan. Yksi näistä algoritmeista on yhdistämisalgoritmi, jolla yhdistetään aikaisempien algoritmien tuottamasta datasta paras data Clock Discipline -algoritmia varten. [9]

3 TOTEUTUS KÄYTTÄEN RTSP-PROTOKOLLAA

Tässä luvussa esitellään toteutus käyttäen RTSP-protokollaa. Tähän toteutukseen päädyttiin, koska RTSP on yleisesti käytetty suoratoistoprotokolla, jota voidaan käyttää RTP:n ja sen lähteiden tahdistuksen kanssa. Suoratoistoon päädyttiin ylipäätään, koska tahdistus on mahdollista toteuttaa sen avulla tarkemmin ja toistettavaa sisältöä on helpompi hallita.

Toteutukseen kuuluu kaksi ohjelmaa, palvelin- ja asiakasohjelma. Palvelinohjelma tarjoilee verkossa oleville laitteille videota, jota laitteiden asiakasohjelmat voivat toistaa. Kummassakin ohjelmassa on käytetty C-ohjelmointikieltä ja sille saatavilla olevaa GStreamer-kirjastoa. Ohjelmissa käytetään esimerkkinä *127.0.0.1*-osoitetta, joka on loopback-osoite. Tämä osoite tulee korvata palvelimen IP-osoitteella, jos palvelin- ja asiakasohjelmaa suoritetaan eri laitteilla.

GStreamer on avoimen lähdekoodin kirjasto, joka sisältää median käsittelyyn sopivia komponentteja. GStreamer tukee ohjelmistoja suoratoistosta videon ja audion miksaamiseen ja epälineaariseen muokkaukseen. [10]

3.1 Palvelinohjelma

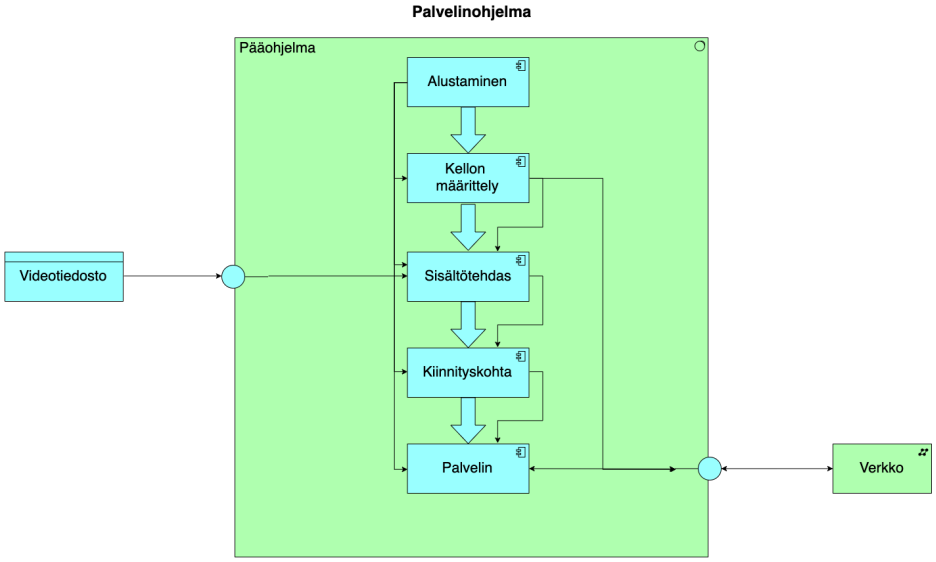
Palvelinohjelman tavoite on suoratoistaa sille annettua mediasisältöä asiakasohjelmille RTSP-protokollaa käyttäen. RTSP-protokolla taas käyttää RTP-protokollaa tiedonsiirtoon, jolloin palvelinohjelma voi hyödyntää sen NTP-aikaleimoja tahdistamiseen. Kuvassa 3.1 on palvelinohjelman rakenne.

GStreameriin on lisäkirjasto *GStreamer RTSP Server*, jossa on toteutettu yleisiä RTSP-palvelimen toteutukseen tarvittavia funktioita. GStreamer RTSP Server -lisäkirjasto vaatii myös lisäosat *GStreamer Plugins Good* ja *GStreamer Plugins Ugly*. Nämä lisäosat lisäävät muun muassa tuen H.264:lla koodatulle videolle. [11]

GStreamer vaatii alustamista ennen käyttöä. Samalla alustetaan myös kaikki vaadittavat muuttujat, joihin tallennetaan ohjelman ajon kannalta oleelliset asiat.

Liitteessä A palvelinohjelma alustetaan koodin osalla

```
1 gst_init (&argc, &argv);
2
3 GstRTSPServer *server;
```



Kuva 3.1. *Palvelinohjelman rakenne.*

```
4 GMainLoop *loop;
5 GstRTSPMediaFactory *factory;
6 GstRTSPMountPoints *mounts;
```

`Server`-muuttuja pitää sisällään itse palvelinolion. `Loop`-muuttuja pitää sisällään palvelusilmukan, joka pitää huolen että palvelinohjelma pysyy päällä ja palvelee yhdistäviä asiakasohjelmia. `Factory`-muuttujaan tallennetaan sisältötehdas (engl. media factory), joka luo mediasisällön siirrettävässä muodossa asiakasohjelmille. `Mounts`-muuttuja sisältää kaikki kiinnityskohdat (engl. mount points) RTSP-palvelimelle. Kiinnityskohdat käyttävät URI-osotteita. Asiakasohjelmat pyytävät palvelinohjelmalta sisältöä näiden kautta. Esimerkiksi toteutuksessa voisi olla useampi sisältötehdas ja jokaisella olisi oma kiinnityskohta, joiden avulla asiakasohjelmat erottavat sisällöt toisistaan.

Palvelinohjelmassa määritetään RTP-protokollan käyttämä kello

```
1 global_clock = gst_system_clock_obtain();
2 gst_net_time_provider_new(global_clock, "0.0.0.0", 8554);
```

Kelloa voidaan käyttää asiakasohjelmien tahdistukseen. Asiakasohjelmassa on oma toteutuksensa kellon löytämiseen palvelimelta. Palvelin esitellään tarkemmin luvussa 3.2.

Palvelinohjelmassa luodaan peruskomponentit valmiiksi alustettuihin muuttujiin:

```
1 loop = g_main_loop_new (NULL, FALSE);
2 server = gst_rtsp_server_new();
3 mounts = gst_rtsp_server_get_mount_points(server);
4 factory = gst_rtsp_media_factory_new();
```

Peruskomponenttien luonnin jälkeen määritellään sisältötehtaan tuottama sisältö:

```
1 gst_rtsp_media_factory_set_launch(factory, "( pushfile://[FILE URL HERE] !
      qtdemux name=d d. ! queue ! rtph264pay pt=96 name=pay0 d. ! queue !
      rtpmp4apay pt=97 name=pay1 )");
```

```
2  gst_rtsp_media_factory_set_shared (factory, TRUE);
3  gst_rtsp_media_factory_set_media_gtype (factory, TEST_RTSP_MEDIA);
```

Parametrin osa *pushfile://* tarkoittaa, että seuraavaksi tulee tiedoston sijainti. Tiedosto toimii siis median lähteenä. Parametrissä *rtph264pay* tarkoittaa, että lähetetään H.264:lla koodattua sisältöä. Lisäksi sisältötehtaasta tehdään jaettu ja sen medialle määritetään tyypiksi *TEST_RTSP_MEDIA*, joka on määritelty aikasemmin liitteessä A. Seuraavaksi lisätään aikaisemmin määritetty kello tahdistamaan sisältöä:

```
1  gst_rtsp_media_factory_set_clock(factory, global_clock);
2  gst_rtsp_mount_points_add_factory(mounts, "/test", factory);
```

Jälkimmäinen koodirivi laittaa sisällön saataville polkuun */test*. Nyt sisältö on toistettavissa esimerkiksi IP-osoitteella *127.0.0.1*, portilla *8554* ja polulla */test*. Eli esimerkiksi VLC-mediasoittimella tai liitteen B ohjelmalla voi avata *rtsp://127.0.0.1:8554/test* -osoitteen.

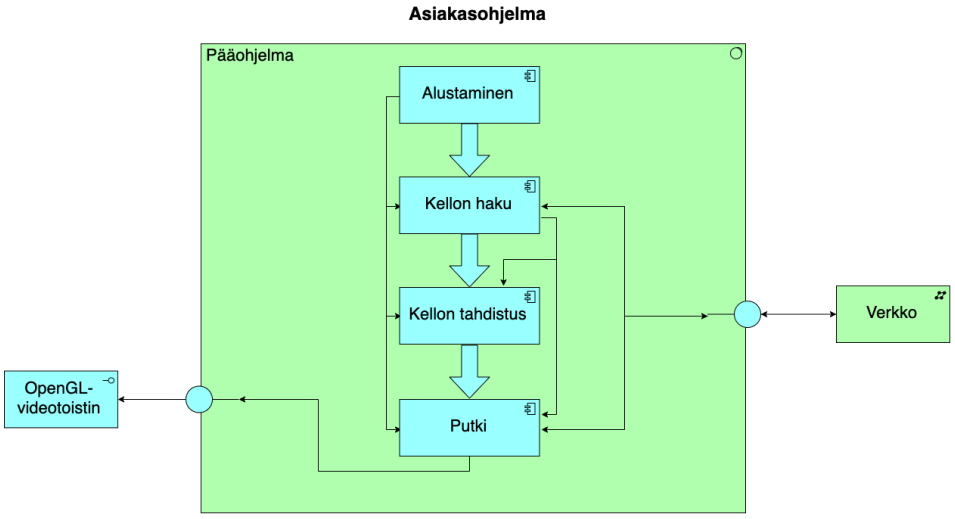
Palvelinohjelma käynnistetään komentorivillä seuraavalla komennolla:

```
./palvelin
```

olettaen, että ohjelma on nimetty *palvelin* nimellä.

3.2 Asiakasohjelma

Asiakasohjelman tavoite on suoratoistaa palvelinohjelman tarjoamaa mediasisältöä siten, että kaikki asiakasohjelmat toistavat sitä samaan aikaan. Asiakasohjelmassa voi lisätä hieman viivettä toistoon, minkä seurauksena ohjelmalla on enemmän mediaa vastaanotettuna. Tällöin toistaminen on tahdistetusti helpompaa, sillä tiedonsiirron viiveen vaihtelu ei vaikuta suuresti sillä hetkellä toistettavaan mediaan. Asiakasohjelma hyödyntää myös samoja GStreamerin lisäkirjastoja kuin palvelinohjelma, *GStreamer RTSP Server* -lisäkirjastoa lukuunottamatta. Kuvassa 3.2 on asiakasohjelman rakenne.



Kuva 3.2. *Asiakasnohjelman rakenne.*

Kuten palvelinohjelmassakin, GStreamer vaatii alustamisen ennen sen käyttöä asiakasohjelmassa. Samalla alustetaan toistoa varten käytettävät muuttujat:

```
GstClock *net_clock;
gchar *server;
gint clock_port;
GstElement *pipe;
GMainLoop *loop;

gst_init(&argc, &argv);
```

`Net_clock`-muuttujaa tullaan käyttämään palvelimelta saadun kellon tallentamiseen. `Server`-muuttuja alustetaan, jotta siihen voidaan tallentaa palvelimen IP-osoite ja `clock_port`-muuttuja taas palvelimen portin tallentamiseen. `Pipe`-muuttujaan tullaan tallentamaan asiakasohjelman putki. `Loop`-muuttujaa tullaan käyttämään ohjelman silmukkana, joka pitää ohjelman päällä.

Seuraavaksi haetaan palvelimen IP-osoite asiakasohjelman ensimmäisestä argumentista:

```
gchar* server_address = argv[1];
gchar* server_ip = malloc(sizeof(argv[1]));
strncpy(server_ip, server_address, strlen(server_address));
server = strtok(server_ip, "rtsp://");
clock_port = 8554;
```

Ensimmäisen argumentin sisältö tallennetaan `server_address`-muuttujaan ja siitä erotellaan IP-osoite. Portti on sovitusti vakiona 8554. Kyseistä IP-osoitetta ja porttia käytetään kellon tahdistamisessa. Asiakasohjelma hakee kellon palvelimelta, jos se on mahdollista:

```
net_clock = gst_net_client_clock_new("net_clock", server, clock_port, 0);
if (net_clock == NULL) {
    g_print ("Failed to create net clock client for %s:%d\n",
             server, clock_port);
    return 1;
}
```

Kellon hakemisen epäonnistuessa tulostetaan virheilmoitus ja lopetetaan ohjelman suorittaminen. Onnistuessa odotetaan, että saadaan kellot tahdistettua:

```
gst_clock_wait_for_sync(net_clock, GST_CLOCK_TIME_NONE);
```

Asiakasohjelma määrittää ohjelman silmukan ja putken, joka suorittaa sisällön käsittelyn:

```
loop = g_main_loop_new(NULL, FALSE);

pipe = gst_element_factory_make("playbin", NULL);
g_object_set(pipe, "uri", server_address, NULL);
g_signal_connect(pipe, "source-setup", G_CALLBACK(source_created), NULL);
```

Putki määritellään käyttämään ohjelman toisen argumentin sisältöä lähteen sijaintina. Sijainti on URI-osoite, joka sijaitsee `server_address`-muuttujassa. Sen sisältö voisi ol-

la *rtsp://127.0.0.1:8554/test*, jota käytetään palvelinohjelmassa. Lähteen käyttöönotossa suoritetaan `source_created`-funktio, joka lisää toistoviiveen lähteeseen. Toistoviivettä käytetään viivyttämään toistoa NTP-lähteen perusteella [12]. Lisäksi `source_created`-funktio määrittää lähteen hyödyntämään NTP:tä tahdistamiseen:

```
1 static void source_created (GstElement *pipe, GstElement *source) {
2     g_object_set(source, "latency", PLAYBACK_DELAY_MS, "ntp-time-source", 3,
       "buffer-mode", 4, "ntp-sync", TRUE, NULL);
3 }
```

Lisäksi asetetaan palvelimelta vastaanotettu kello tahdistusta varten, sekä asetetaan vakioviive:

```
1 gst_pipeline_use_clock(GST_PIPELINE(pipe), net_clock);
2 gst_pipeline_set_latency(GST_PIPELINE(pipe), 500*GST_MSECOND);
```

Vakioviive on putken viive, joka on käytössä kun tahdistuksen toistoviive ei ole käytössä [13].

Asiakasnohjelma käynnistetään komentorivillä seuraavalla komennolla:

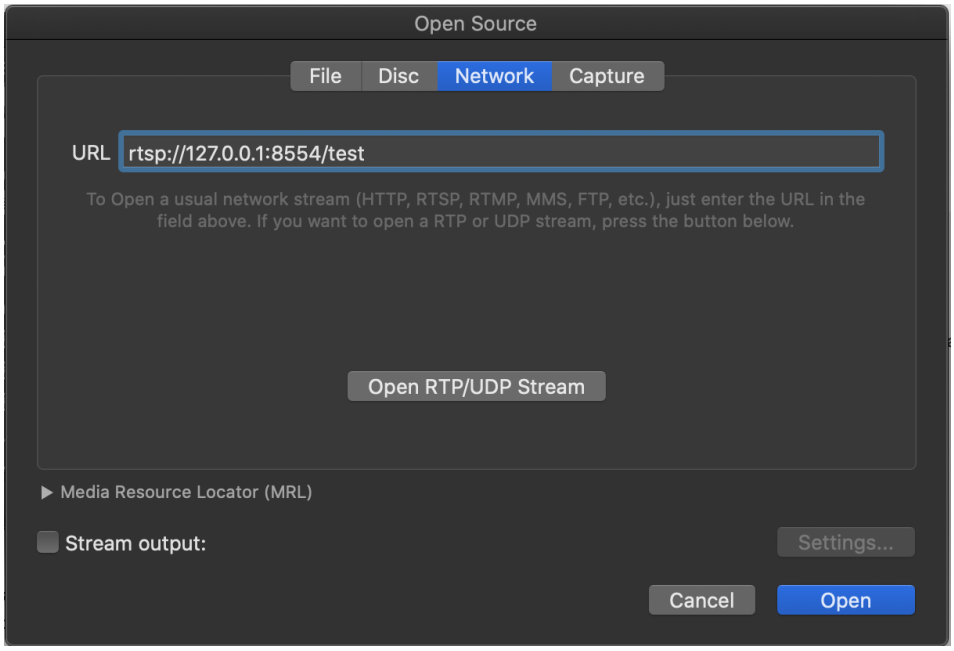
```
./asiakas rtsp://127.0.0.1:8554/test
```

olettaen, että ohjelma on nimetty *asiakas* nimellä. IP-osoitte tulee vaihtaa, jos palvelinta ei suoriteta samalla laitteella.

3.3 VLC-mediasoitin

VLC-mediasoitin on VideoLAN-organisaation tuottama avoimen lähdekoodin mediasoitin. VLC tukee useita tapoja toistaa mediasisältöä, kuten suoratoistaa muun muassa RTSP-palvelimelta. Lisäksi sillä voi toistaa useilla koodekeilla koodattuja tiedostoja. VLC tukee myös yleisimpiä Linux-käyttöjärjestelmiä, Windowsia, macOS:ää, iOS:ää ja Androidia. [14]

Kuvassa 3.3 yhdistetään VLC:llä paikalliseen palvelinohjelmaan käyttäen RTSP-protokollaa. Tässä siis käytetään VLC:tä luvussa 3.2 määritellyn asiakasohjelman sijaan. Asiakasohjelmaan verrattuna VLC varastoi välimuistiin ennalta määrätyn mittaisen osan suoratoistettavasta mediasta ennen toistoa, sen sijaan että se tarvittaessa lisäisi toistoon viivettä. VLC:ssä tästä käytetään englanninkielistä termiä caching. Sen tulisi tuottaa erilaisia tuloksia verrattuna asiakasohjelman toteutukseen.



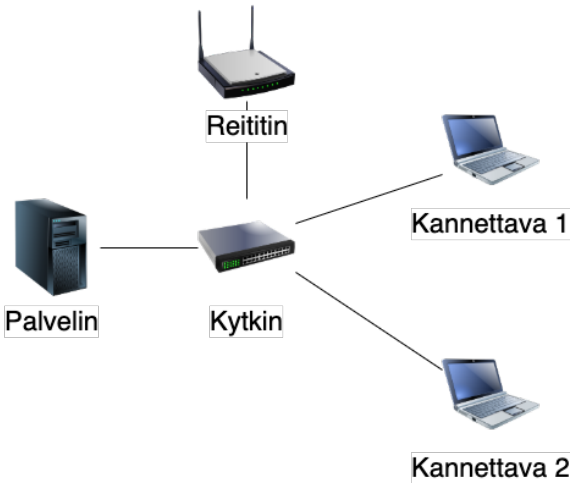
Kuva 3.3. *VLC-esimerkki yhdistämisestä paikalliseen, liiteen A mukaiseen palvelinohjelmaan.*

4 TAHDISTUKSEN TESTAUS TOTEUTETUSSA JÄRJESTELMÄSSÄ

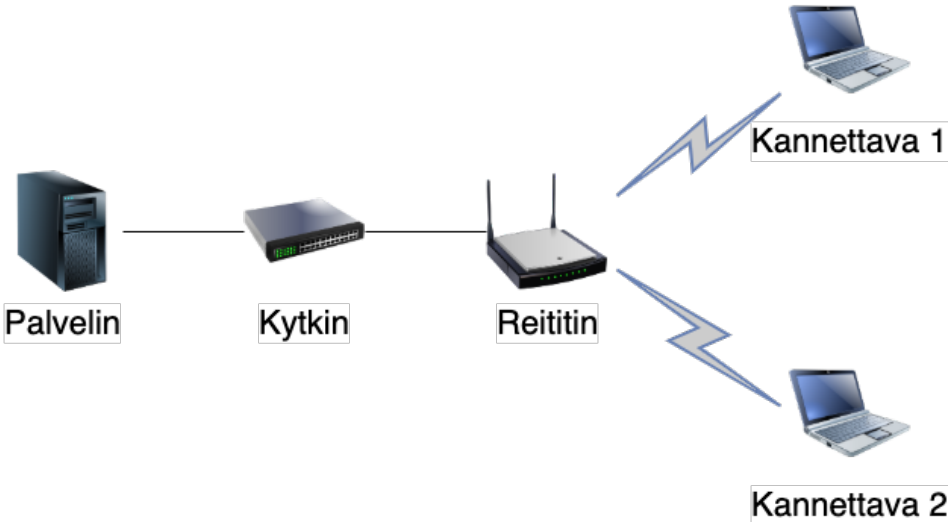
Tässä luvussa käydään läpi tahdistuksen testaus ja sen tulokset. Luvussa 4.1 on esitetty langallisessa lähiverkossa saadut tulokset ja luvussa 4.2 on esitetty langattomassa lähiverkossa saadut tulokset. Luvussa 4.3 analysoidaan edellisten tulosten eroja, ja mitä ne tarkoittavat.

Tahdistuksen testaamiseen käytetään kahta kannettavaa tietokonetta, koska ne on helppo asetella vierekkäin ja niissä on testiin tarvittavat välineet, kuten näyttö, verkkokortti ja itse tietokone. Asiakasohjelmaa ajetaan kannettaville luoduissa Linux-virtuaalikoneissa, sillä GStreamer-kirjasto on helpommin asennettavissa Linuxille kuin Windowsille. Kannettavilla tehdään useampi testi, joissa vaihdetaan toistoviivettä ja verkon rakennetta. Toistoviive on ensimmäisessä testissä 0 millisekuntia, toisessa 40 millisekuntia ja kolmannessa 1000 millisekuntia. Samat testit suoritetaan WLAN-verkon ja LAN-verkon avulla. Lisäksi testataan VLC:tä asiakasohjelman verrokkina. Verkot toteutetaan kuvien 4.1 ja 4.2 mukaisesti. Käytössä olevasta verkosta kannettavat voivat yhdistää palvelimelle, josta testivideo suoratoistetaan.

WLAN-verkkoa käyttäen voidaan tuoda järjestelmään hieman epävarmuutta, sillä WLAN-verkoissa tiedonsiirron viive voi vaihdella suuresti olosuhteiden mukaan. Tällöin voidaan testata langattoman yhteyden todellista vaikutusta ja saadaan selville järjestelmän luotettavuutta todellisessa käyttötilanteessa, jossa tarvitaan langatonta yhteyttä.

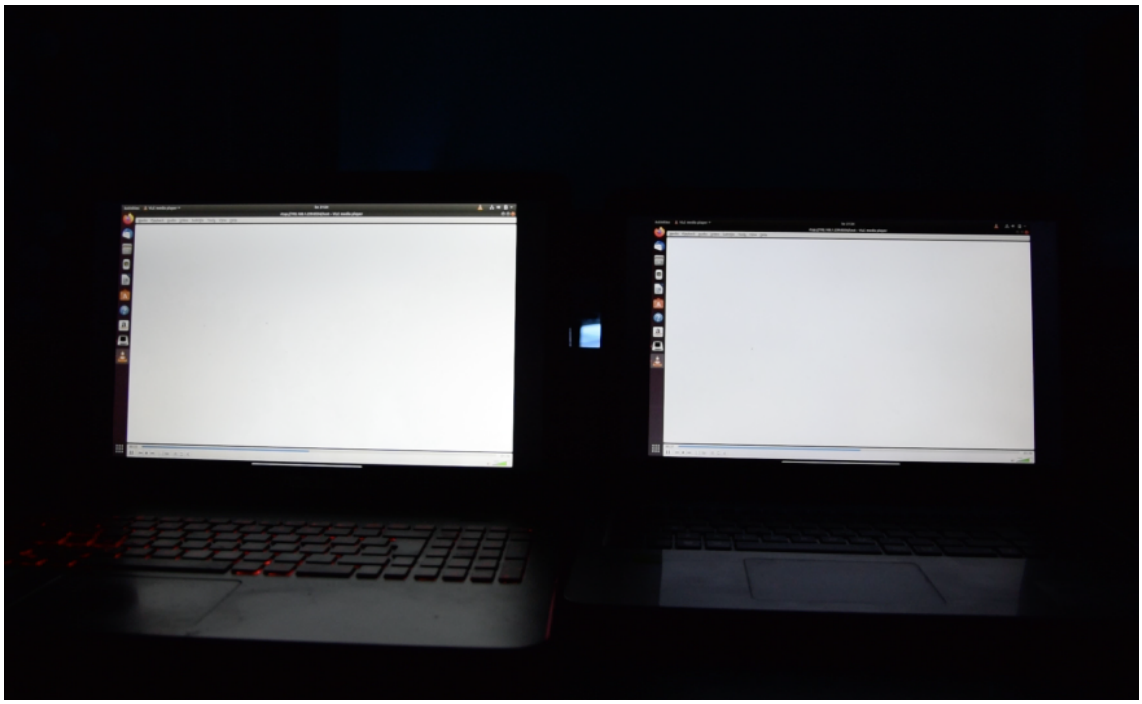


Kuva 4.1. *Testiverkon rakenne, kun testataan LAN-verkon avulla.*



Kuva 4.2. *Testiverkon rakenne, kun testataan WLAN-verkon avulla.*

Testauksen sisältönä käytetään videota, jossa on tietyn väliajoin vaihtuva väri. Tällöin videon tila on helppo huomata ja sitä on myös helppo verrata viereisellä kannettavalla toistettavaan videoon. Testaus tallennetaan kameralla siten, että kuvassa näkyy kummankin kannettavan näyttö ja koko testi tallennetaan videona, kuten on esitetty kuvassa 4.3. Videossa on 60 kuvaa sekunnissa, eli värin vaihtumisessa päästään 1/60 sekunnin tarkkuuteen.



Kuva 4.3. *Kameran näkökulma testauksessa.*

Kuva 4.3 on kameralla kuvatusta videosta yksi kuva. Vasemmalla kannettava 2 ja oikealla kannettava 1, johon verrataan kannettavaa 1.

Testauksen tulokset lasketaan sen perusteella, kuinka kauan kestää, että kummatkin kan-

nettava ovat vaihtaneet ruutunsa väriä. Kannettava 1 valitaan vertailun kohteeksi. Testauksessa kuvattua videota analysoidessa huomattiin, että värin vaihtuessa koko näytön väri ei aina vaihdu kerralla. Keskeneräiset värinvaihdokset saattavat johtua kannettavissa käytetyistä virtuaalikoneista, toisto-ohjelmistoista tai testivideosta. Värien vaihtumisnopeus kuitenkin vaihteli, joten voidaan olettaa, että se ei johtunut itse testivideosta.

4.1 Tulokset langallisessa lähiverkossa

Testiverkko toteutettiin kuvan 4.1 mukaisesti. Taulukossa 4.1 on mitattu aikaero värin vaihtuessa LAN-verkon testeistä. Tulokset ovat merkitty sekunneissa ja pyöristetty kahden desimaalin tarkkuuteen. Vertailun kohteena on kannettava 1 eli negatiiviset arvot tarkoittavat kannettavan 2 vaihtaneen väriä ennen kannettavaa 1. Positiivisissa arvoissa kannettava 1 vaihtoi väriä ensin. Taulukon *viive*-sarake viittaa luvussa 3.2 määritettyyn toistoviiveeseen. Toistoviivettä vaihdettiin testeissä 0–1000 millisekuntia välillä.

Suurimmillaan toinen kannettava oli testin alkaessa 2,17 sekuntia jäljessä, mutta testin loppua kohden aikaero pieneni. VLC:n testissä laitteiden aikaero oli vain 0–200 millisekuntia. Asiakasohjelman testistä 40 ms toistoviiveellä jäi viimeiset 4 tulosta puuttumaan kameran kuvausongelmien takia.

Ohjelma	Viive (ms)	Vaihto 1	Vaihto 2	Vaihto 3	Vaihto 4	Vaihto 5	Vaihto 6
Asiakasohjelma	0	2,02	1,43	0,13	0,73	0,37	0,00
Asiakasohjelma	40	2,17	1,73	1,50	0,97	0,67	0,23
Asiakasohjelma	1000	1,60	1,17	0,97	0,47	0,03	0,33
VLC	-	0,00	0,07	0,03	0,10	0,07	0,07

Ohjelma	Viive (ms)	Vaihto 7	Vaihto 8	Vaihto 9	Vaihto 10	Vaihto 11	Vaihto 12
Asiakasohjelma	0	-0,20	-0,50	-0,17	-0,57	-0,43	-0,77
Asiakasohjelma	40	1,07	0,80	-	-	-	-
Asiakasohjelma	1000	0,17	0,00	-0,10	0,07	-0,43	-0,80
VLC	-	0,07	0,03	0,03	0,20	0,12	0,07

Taulukko 4.1. *LAN-verkkossa mitattuja aikaeroja pyöristettynä.*

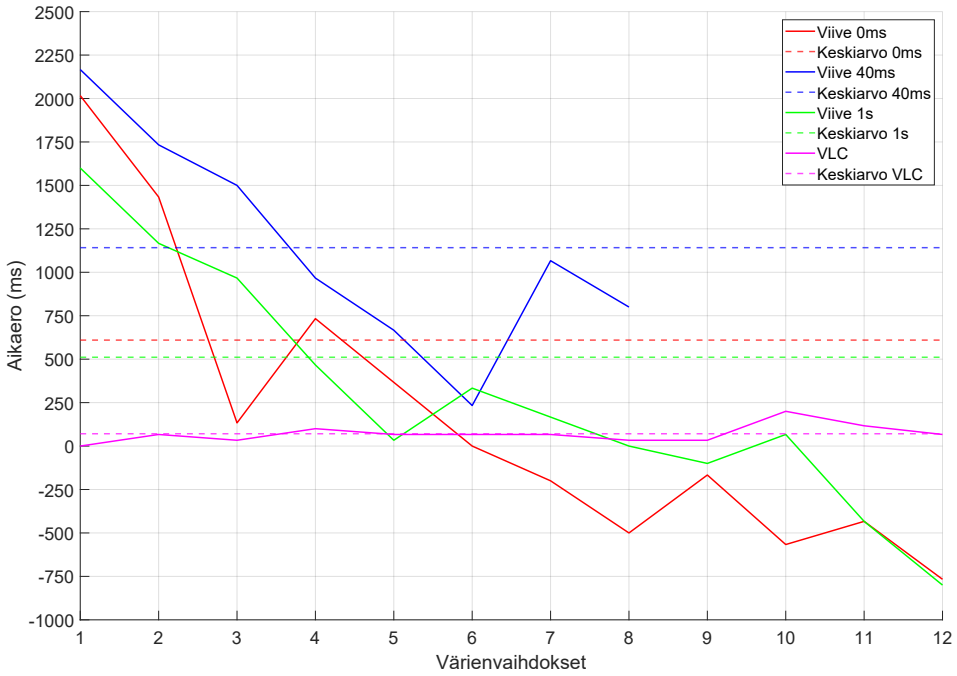
Taulukossa 4.2 on laskettu taulukon 4.1 tulosten itseisarvojen perusteella LAN-testien varianssi ja keskiarvo. Itseisarvoa on käytetty, jotta saadaan taulukon 4.1 tuloksista keskiarvo aikaerolle, joka on riippumaton siitä kumpi kannettava vaihtoi väriä ensin. Varianssin suuruudesta voidaan huomata, että testissä, jossa käytettiin luvun 3.2 asiakasohjelmaa 40 millisekunnin toistoviiveellä, aikaerot vaihtelivat enemmän, kuin muissa LAN-verkolla tehdyissä testeissä. Siinä oli myös suurin aikaero keskiarvon perusteella. Pienin aikaero oli niiden perusteella VLC:llä testattaessa. VLC:n testissä oli myös pienin varianssi.

Kuvassa 4.4 on visualisoitu taulukon 4.1 tulokset ja taulukon 4.2 keskiarvo. Lukuunottamatta testiä 40 millisekuntin toistoviiveellä, jossa osa tuloksista jäi puuttumaan, toistoviive näyttäisi LAN-verkoissa vaikuttavan aikaeron keskiarvoon. Toistoviiveen vaikutuksen voisi vahvistaa, jos testit tehtäisiin uudelleen useaan kertaan. Kuvaajasta selviää, että asia-

Ohjelma	Viive (ms)	Varianssi (ms^2)	Keskiarvo (ms)
Asiakasohjelma	0	342200	609,7
Asiakasohjelma	40	390400	1141,7
Asiakasohjelma	1000	265500	511,1
VLC	-	2600	70,8

Taulukko 4.2. *LAN-verkossa mitattujen tulosten varianssi ja keskiarvo.*

kasohjelma 1000 millisekunnin toistoviiveellä pysyy jokseenkin parhaiten tahdistettuna, verrattuna asiakasohjelmaan muilla toistoviiveillä.



Kuva 4.4. *Taulukkojen 4.1 ja 4.2 LAN-verkossa mitatut tulokset visualisoituna.*

4.2 Tulokset langattomassa lähiverkossa

Taulukossa 4.3 on mittaukset WLAN-verkon testeistä. Verkko toteutettiin kuvan 4.2 mukaisesti. Tulokset ovat merkitty sekunneissa ja pyöristetty kahden desimaalin tarkkuuteen. Tulokset ovat hyvin saman tyylisiä kuin luvussa 4.1, jossa oli LAN-verkon testien tulokset.

WLAN-testeissä aikaerot vaihtelevat huomattavasti, mikä johtuu langattomille verkoille tyypillisestä suorituskyvyn vaihtelusta kanavaolosuhteiden mukaan. Suurimmillaan aikaero oli 1,90 sekuntia testin alussa. VLC:n testissä aikaerot olivat 0–270 millisekuntia.

Taulukossa 4.4 on laskettu taulukon 4.3 tulosten itseisarvojen perusteella WLAN-testien varianssi ja keskiarvo. Itseisarvoa on käytetty, jotta saadaan taulukon 4.3 tuloksista keskiarvo aikaerolle, joka on riippumaton siitä kumpi kannettava vaihtoi väriä ensin. Varianssi

Ohjelma	Viive (ms)	Vaihto 1	Vaihto 2	Vaihto 3	Vaihto 4	Vaihto 5	Vaihto 6
Asiakasohjelma	0	1,33	1,23	0,70	0,43	0,33	1,00
Asiakasohjelma	40	1,13	0,67	0,37	-0,10	-0,37	0,03
Asiakasohjelma	1000	-1,73	-1,90	-0,47	1,30	1,10	0,50
VLC	-	0,13	0,03	0,13	0,07	0,27	0,00

Ohjelma	Viive (ms)	Vaihto 7	Vaihto 8	Vaihto 9	Vaihto 10	Vaihto 11	Vaihto 12
Asiakasohjelma	0	0,50	0,33	0,03	-0,37	-0,63	-1,07
Asiakasohjelma	40	-0,20	-0,53	-1,00	-1,30	-0,03	1,73
Asiakasohjelma	1000	0,90	0,73	0,23	-0,13	-0,53	-0,90
VLC	-	0,23	0,03	0,27	0,07	0,10	0,00

Taulukko 4.3. *WLAN-verkkossa mitattuja aikaeroja.*

on suurimmillaan käyttäen luvun 3.2 asiakasohjelmaa 1000 millisekunnin toistoviiveellä, eli sitä käyttäen kannettavien aikaerot vaihtelivat eniten. VLC:n varianssi oli pienin.

Ohjelma	Viive (ms)	Varianssi (ms^2)	Keskiarvo (ms)
Asiakasohjelma	0	166800	663,9
Asiakasohjelma	40	307500	622,2
Asiakasohjelma	1000	311400	869,4
VLC	-	9600	111,1

Taulukko 4.4. *WLAN-verkossa mitattujen tulosten varianssi ja keskiarvo.*

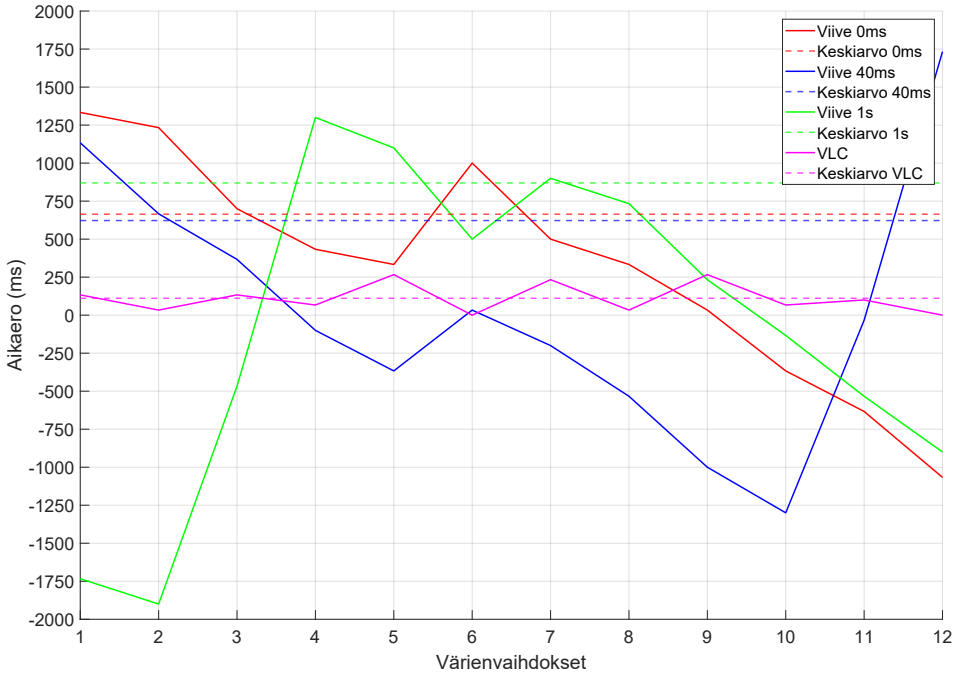
Kuvassa 4.5 on visualisoitu taulukon 4.3 tulokset ja taulukon 4.4 keskiarvo. WLAN-verkoissa toistoviiveen pidentäminen vaikuttaa pienentävän keskiarvoa, kunhan toistoviivettä ei pidennetä liikaa. Toistoviiveen vaikutuksen voisi vahvistaa suorittamalla samat testit uudelleen useaan kertaan.

4.3 Tulosten analysointi

Taulukkojen 4.1 ja 4.3 tulokset vaikuttavat siltä, että ainakin VLC:n caching-ominaisuus vaikuttaisi tuloksiin. Niihin saattavat vaikuttaa myös VLC:n muut ominaisuudet, jotka saattaisivat selvitä sen lähdekoodia tutkimalla.

Testeissä kävi ilmi, että luvussa 3.2 määritellyn asiakasohjelman tahdistus huojuu. Asiakasohjelmaa käyttäessä kannettavat aloittavat eri tahdissa ja pääsevät hetkellisesti samaan tahtiin. Lopuksi kannettavat olivat kuitenkin eri tahdissa, mutta tällä kertaa toinen jäljessä.

VLC:n voidaan huomata korjaavan tahdistustaan tehokkaammin kuin luvun 3.2 asiakasohjelma. Sekä LAN- että WLAN-verkoissa VLC:n aikaerot ovat huomattavasti pienemmät kuin asiakasohjelmalla. Vaikkakin joidenkin värinvaihtojen kohdalla WLAN-verkon tapauksessa VLC:llä on enemmän aikaeroa kuin jollain muulla testillä, VLC on yleisesti



Kuva 4.5. *Taulukkojen 4.3 ja 4.4 WLAN-verkossa mitatut tulokset visualisoituna.*

paremmin tahdistettu.

Taulukossa 4.5 on testien varianssin ja keskiarvon muutokset, kun siirrytään LAN-verkosta WLAN-verkkoon. Asiakasohjelman varianssi laski 0:n ja 40 millisekunnin toistoviiveillä, mutta vain 40 millisekunnin toistoviiveellä keskiarvo laski huomattavasti. Tästä voimme päätellä, että asiakasohjelman testi 40 millisekunnin toistoviiveellä LAN-verkossa antoi mahdollisesti liian suuria tuloksia, mikä saattoi johtua verkon ruuhkaisuudesta, sillä verkossa kulki muutakin liikennettä.

Ohjelma	Viive (ms)	Varianssi (ms^2)	Keskiarvo (ms)
Asiakasohjelma	0	-175500	54,2
Asiakasohjelma	40	-82900	-519,4
Asiakasohjelma	1000	45900	358,3
VLC	-	6900	40,3

Taulukko 4.5. *LAN-verkosta siirtyminen WLAN-verkkoihin. Mitattujen tulosten varianssin ja keskiarvon muutokset.*

Taulukko 4.5 pääasiallisesti tukee ajatusta WLAN-verkon huonommasta suorituskyvystä, sillä suuri osa aikaeroihin viittaavista arvoista nousi WLAN-verkon testeissä. Joten LAN-verkkoa kannattaa käyttää tahdistuksen parantamiseksi, mutta WLAN-verkkokaan ei ole huono vaihtoehto, jos sen voi toteuttaa helpommin ja vastaanottavat ohjelmistot saadaan optimoitua. Ohjelmistojen optimoinnissa tulee ottaa huomioon caching, joka voi olla suuresti avuksi. VLC luultavasti käy ratkaisuksi, jos toteutusta ei häiritse sen ylimääräiset ominaisuudet. VLC on toisaalta todella hyvä siinä mielessä, että se tukee todella paljon eri koodekkeja ja tapoja toistaa tai suoratoistaa sisältöä. Palvelinohjelmalla ei luultavasti

ole juurikaan merkitystä, kunhan se tukee haluttuja protokollia, koodekkeja ja tahdistusta, kuten luvussa 3.1 toteutettu palvelinohjelma.

5 YHTEENVETO

Audion ja videon suoratoistamiseen on useita protokollia, jotka ovat osa hieman eritarkoituksiin. Audion ja videon pakallisen toiston tahdistamiseen teoriassa voi käyttää NTP:tä, mutta tahdistusta ei ole helppo korjata kesken toiston. Testitoteutukseen valittiin RTSP-protokolla, joka on varsin toimiva suoratoistoprotokolla sen tukeutuessa RTP-protokollaan ja sen käyttämään lähteiden tahdistukseen. Lähteiden tahdistuksen käyttäminen vastaanottajien tahdistukseen toimi kohtalaisen hyvin, jos muutaman sadan millisekunnin aikaero ei häiritse. VLC:n caching- ja muut ominaisuudet tekevät siitä hyvän asiakasohjelman, joka tukee monia protokollia ja koodekkeja jo valmiiksi, joten toimiva toteutus ei vaatisi omaa asiakasohjelmistoaan, jos VLC:n ylimääräiset paikallisen toiston ominaisuudet eivät haittaa suorituskyvyllisesti.

Tulevaisuudessa voisi tehdä testit uudelleen useamman kerran, sillä sitten voitaisiin oikeasti selvittää GStreamer-kirjaston tukeman toistoviiveen vaikutus tahdistukseen. Testit voisi myös tehdä verkossa, jossa ei liiku ylimääräistä liikennettä. Lisäksi VLC:n caching-ominaisuuden ajan pituutta voitaisiin vaihdella ja testata sen vaikutusta.

LÄHTEET

[1] Nurrohman, A. ja Abdurohman, M. High Performance Streaming Based on H264 and Real Time Messaging Protocol (RTMP). *2018 6th International Conference on Information and Communication Technology (ICoICT)*. Toukokuu 2018, pp. 174—177. DOI: 10.1109/ICoICT.2018.8528770.

[2] Schulzrinne, H., Casner, S., Frederick, R. ja Jacobson, V. *RTP: A Transport Protocol for Real-Time Applications*. Network Working Group. 2003, 89 p. DOI: 10.17487/RFC3550.

[3] *WebRTC: Web Real-Time Communication*. 2017. URL: https://webrtc.org (viitattu 05. 10. 2019).

[4] Nurminen, J. K., Meyn, A. J. R., Jalonen, E., Raivio, Y. ja Garcıa Marrero, R. P2P media streaming with HTML5 and WebRTC. *2013 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. Huhtikuu 2013, pp. 63—64. DOI: 10.1109/INFCOMW.2013.6970739.

[5] Schulzrinne, H., Rao, A., Lanphier, R., Westerlund, M. ja Stiemerling, M. *Real-Time Streaming Protocol Version 2.0*. Internet Engineering Task Force (IETF). 2016. DOI: 10.17487/RFC7826.

[6] Parmar, H. ja Thornburgh, H. *Adobe's Real Time Messaging Protocol*. Adobe. 2012, 52 p. URL: http://wwwimages.adobe.com/www.adobe.com/content/dam/acom/en/devnet/rtmp/pdf/rtmp_specification_1.0.pdf.

[7] Pantos, R. ja May, W. *HTTP Live Streaming*. Apple Inc. 2017, 60 p. DOI: 10.17487/RFC8216.

[8] *Information technology — Dynamic adaptive streaming over HTTP (DASH) — Part 1: Media presentation description and segment formats*. ISO/IEC, 2019, 225p. URL: https://www.iso.org/standard/75485.html.

[9] Mills, D., Delaware, U., Martin, J., Burbank, J. ja Kasch, W. *Network Time Protocol Version 4: Protocol and Algorithms Specification*. Internet Engineering Task Force (IETF). 2010, 110 p. DOI: 10.17487/RFC5905.

[10] *Gstreamer - open source multimedia framework*. 2019. URL: https://gstreamer.freedesktop.org/ (viitattu 08. 12. 2019).

[11] *GStreamer Documentation - GStreamer RTSP Server*. URL: https://gstreamer.freedesktop.org/documentation/gst-rtsp-server/index.html?gi-language=c (viitattu 08. 12. 2019).

[12] *GStreamer Documentation - Clocks and synchronization in GStreamer*. URL: https://gstreamer.freedesktop.org/documentation/application-development/advanced/clocks.html?gi-language=c (viitattu 04. 12. 2019).

[13] *GStreamer Documentation - GstPipeline*. URL: https://gstreamer.freedesktop.org/documentation/gstreamer/gstpipeline.html?gi-language=c (viitattu 20. 12. 2019).

[14] *VLC media player*. 2019. URL: https://www.videolan.org/vlc/ (viitattu 28. 11. 2019).

LIITE A PALVELINOHJELMA

```
#include <stdio.h>
#include <gst/gst.h>
#include <gst/rtsp-server/rtsp-server.h>
#include <gst/net/gstnettimeprovider.h>

GstClock *global_clock;

#define TEST_TYPE_RTSP_MEDIA_FACTORY       (test_rtsp_media_factory_get_type
    ())
#define TEST_TYPE_RTSP_MEDIA               (test_rtsp_media_get_type ())

GType test_rtsp_media_get_type(void);

typedef struct TestRTSPMediaClass TestRTSPMediaClass;
typedef struct TestRTSPMedia TestRTSPMedia;

struct TestRTSPMediaClass {
    GstRTSPMediaClass parent;
};

struct TestRTSPMedia {
    GstRTSPMedia parent;
};

static gboolean custom_setup_rtpbin(GstRTSPMedia *media, GstElement *rtpbin)
    ;

G_DEFINE_TYPE (TestRTSPMedia, test_rtsp_media, GST_TYPE_RTSP_MEDIA);

static void test_rtsp_media_class_init (TestRTSPMediaClass *test_klass) {
    GstRTSPMediaClass *klass = (GstRTSPMediaClass *) (test_klass);
    klass->setup_rtpbin = custom_setup_rtpbin;
}

static void test_rtsp_media_init (TestRTSPMedia *media) {}

// Määritetään tahdistus
static gboolean custom_setup_rtpbin (GstRTSPMedia *media, GstElement *rtpbin
    ) {
    g_object_set(rtpbin, "ntp-time-source", 3, NULL);
    return TRUE;
}
```

```
40
41  // Pääohjelma
42  int main(int argc, char *argv[]) {
43
44      // Alustetaan
45      gst_init (&argc, &argv);
46
47      GstRTSPServer *server;
48      GMainLoop *loop;
49      GstRTSPMediaFactory *factory;
50      GstRTSPMountPoints *mounts;
51
52      loop = g_main_loop_new(NULL, FALSE);
53
54      // Luodaan palvelimen kello
55      global_clock = gst_system_clock_obtain();
56      gst_net_time_provider_new(global_clock, "0.0.0.0", 8554);
57
58      // Määritetään palvelin, "kiinnityskohdat"ja "mediatehdas"
59      server = gst_rtsp_server_new();
60
61      mounts = gst_rtsp_server_get_mount_points(server);
62
63      factory = gst_rtsp_media_factory_new();
64
65      // Määritellään mediatehdas siten, että se tarjoaa tiedoston h.264 koodauksella
66      gst_rtsp_media_factory_set_launch(factory, "( pushfile://[FILE URL HERE]
         ! qtdemux name=d d. ! queue ! rtph264pay pt=96 name=pay0 d. ! queue !
        rtpmp4apay pt=97 name=pay1 )");
67      gst_rtsp_media_factory_set_shared (factory, TRUE);
68      gst_rtsp_media_factory_set_media_gtype (factory, TEST_TYPE_RTSP_MEDIA);
69
70      // Määritellään mediatehtaalle kello
71      gst_rtsp_media_factory_set_clock(factory, global_clock);
72
73      // Kiinnitetään mediatehdas "/test-polkuun.
74      gst_rtsp_mount_points_add_factory(mounts, "/test", factory);
75
76
77      g_object_unref(mounts);
78
79
80      gst_rtsp_server_attach(server, NULL);
81
82      // Palvelin valmiina palvelemaan
83      g_print ("stream ready at rtsp://127.0.0.1:8554/test\n");
84      g_main_loop_run(loop);
85
86      return 0;
87  }
```

LIITE B ASIAKASOHJELMA

```
#include <stdlib.h>
#include <gst/gst.h>
#include <gst/net/gstnet.h>

// Asetetaan toistoviive
#define PLAYBACK_DELAY_MS 40

// Asetetaan lähteelle toistoviive ja määritetään sille tahdistus
static void source_created (GstElement *pipe, GstElement *source) {
    g_object_set(source, "latency", PLAYBACK_DELAY_MS, "ntp-time-source", 3,
     "buffer-mode", 4, "ntp-sync", TRUE, NULL);
}

// Käsitellään ilmenevät virhe viestit ja videon päättymis viesti
static gboolean message(GstBus *bus, GstMessage *message, gpointer user_data
    ) {
    GMainLoop *loop = user_data;
    switch (GST_MESSAGE_TYPE(message)) {
        case GST_MESSAGE_ERROR: {
            GError *err = NULL;
            gchar *name, *debug = NULL;

            name = gst_object_get_path_string(message->src);
            gst_message_parse_error(message, &err, &debug);

            // Tulostetaan virheviesti komentoriville
            g_printerr("ERROR: from element %s: %s\n", name, err->message);
            if(debug != NULL)
                g_printerr("Additional debug info:\n%s\n", debug);

            g_error_free(err);
            g_free(debug);
            g_free(name);

            g_main_loop_quit(loop);
            break;
        }
        case GST_MESSAGE_EOS:
            // Tulostetaan videon päättymisviesti komentoriville
            g_print("Got EOS\n");
            g_main_loop_quit(loop);
            break;
```

```
41          default:
42              break;
43      }
44      return TRUE;
45  }
46
47  // Pääohjelma
48  int main(int argc, char *argv[]) {
49
50      // Alustetaan ohjelma
51      GstClock *net_clock;
52      gchar *server;
53      gint clock_port;
54      GstElement *pipe;
55      GMainLoop *loop;
56
57      gst_init(&argc, &argv);
58
59      // Parsitaan palvelimen osoite argumentista 1
60      gchar* server_address = argv[1];
61      gchar* server_ip = malloc(sizeof(argv[1]));
62      strncpy(server_ip, server_address, strlen(server_address));
63      server = strtok(server_ip, "rtsp://");
64      clock_port = 8554;
65
66      // Tahdistetaan palvelimen kelloon
67      net_clock = gst_net_client_clock_new("net_clock", server, clock_port, 0)
        ;
68      if (net_clock == NULL) {
69          g_print ("Failed to create net clock client for %s:%d\n",
70                   server, clock_port);
71          return 1;
72      }
73
74      gst_clock_wait_for_sync(net_clock, GST_CLOCK_TIME_NONE);
75
76      loop = g_main_loop_new(NULL, FALSE);
77
78      // Määritellään "toistoputki"
79      pipe = gst_element_factory_make("playbin", NULL);
80      g_object_set(pipe, "uri", server_address, NULL);
81      g_signal_connect(pipe, "source-setup", G_CALLBACK(source_created), NULL)
        ;
82
83      // Määritellään toistoputki käyttämään tahdistettua kelloa
84      gst_pipeline_use_clock(GST_PIPELINE(pipe), net_clock);
85
86      gst_pipeline_set_latency(GST_PIPELINE(pipe), 500*GST_MSECOND);
87
88      // Aloita sisällön toistaminen
89      if (gst_element_set_state (pipe, GST_STATE_PLAYING) ==
       GST_STATE_CHANGE_FAILURE) {
```

```
90          g_print ("Failed to set state to PLAYING\n");
91          goto exit;
92      }
93
94      // Käsittele toistoputkesta tulevat viestit
95      gst_bus_add_signal_watch(GST_ELEMENT_BUS(pipe));
96      g_signal_connect(GST_ELEMENT_BUS(pipe), "message", G_CALLBACK(message),
        loop);
97
98      g_main_loop_run(loop);
99
100     // Siivoa ohjelma ennen lopetusta
101     exit:
102     gst_element_set_state(pipe, GST_STATE_NULL);
103     gst_object_unref(pipe);
104     g_main_loop_unref(loop);
105
106     return 0;
107 }
```